LLNL-CONF-595032

NUMERICAL REPRODUCIBILITY FOR IMPLICIT MONTE CARLO SIMULATIONS

M. Cleveland, T. Brunner, N. Gentile

October 24, 2012

The International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering
Sun Valley, ID, United States
May 5, 2013 through May 9, 2013

International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013)
Sun Valley, Idaho, USA, May 5-9, 2013, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2013)

NUMERICAL REPRODUCIBILITY FOR IMPLICIT MONTE CARLO SIMULATIONS

Mathew Cleveland, Thomas Brunner, and Nick Gentile [*]
Lawrence Livermore National Laboratory
L-405, P. O. Box 808, Livermore California 94550
cleveland7@llnl.gov; brunner6@llnl.gov; gentile1@llnl.gov;

ABSTRACT

We describe and compare different approaches for achieving numerical reproducibility in photon Monte Carlo simulations. Reproducibility is desirable for code verification, testing, and debugging. Parallelism creates a unique problem for achieving reproducibility in Monte Carlo simulations because it changes the order in which values are summed. This is a numerical problem because double precision arithmetic is not associative.

In [1], a way of eliminating this roundoff error using integer tallies was described. This approach successfully achieves reproducibility at the cost of lost accuracy by rounding double precision numbers to fewer significant digits. This integer approach, and other extended reproducibility techniques, are described and compared in this work.

Increased precision alone is not enough to ensure reproducibility of photon Monte Carlo simulations. A non-arbitrary precision approaches required a varying degree of rounding to achieve reproducibility. For the problems investigated in this work double precision global accuracy was achievable by using 100 bits of precision or greater on all unordered sums which where subsequently rounded to double precision at the end of every time-step.

Key Words: Implicit Monte Carlo, Numerical Reproducibility, Extended Precision

1. INTRODUCTION

There are three main issues that can cause a Monte Carlo code to get different results when run on different numbers of domains. The first is that domain decomposition can cause the code to use a different sequence of pseudo-random numbers. This can be dealt with by assigning each particle its own random number generator seed [1]. The second is that the order of operations on floating point numbers can change, leading to different results. In other words, double precision arithmetic is not associative: $a + (b + c) \neq (a + b) + c$. The third and final problem is that domain decomposition can result in slightly different mesh properties (cell sizes, distance to boundaries, etc..) causing deviations in problem solutions. The second and third problems described here also apply to deterministic codes.

1.1. Overview of physics algorithm

This work has been done in the context of an Implicit Monte Carlo (IMC) code. We will begin by briefly describe that algorithm. Details of the algorithm can be found in [2]. The algorithm simulates the time-dependent interaction of photons and matter. It does this by creating, tracking, and destroying particles whose behavior models that of real photons in matter. This requires calculating probabilities for physical

[*]LLNL-CONF-595032

events such as emission and scattering. The behavior of each photon is determined by using a pseudo-random number to pick one of these physical events and simulate it. This is repeated until the photon is completely absorbed by the matter, leaves the domain, or reaches the end of the time step.

The behavior of matter is simulated on a grid of zones, each with different material properties, such as temperature and opacity. These zones lose energy when they emit particles, and gain energy when particles pass through them. Particles can visit a zone more than once (for example, by leaving and scattering back in from another zone.) In that case, a particle will deposit energy in the zone more than once.

The Implicit Monte Carlo program used in this work is part of the KULL [3] and ALEGRA [4] inertial confinement fusion simulation codes. Parallel domain decomposition was accomplished using the algorithm described in [5].

There are three main sums in a photon Monte Carlo simulation that require associative addition for reproducibility: the census energy, the source energy, and the energy absorbed by the material. In a radiation hydrodynamics simulation, the momentum absorbed will also need to be added associatively to enable the hydrodynamics simulation to be reproducible. We require associative addition only for the few variables that effect the reproducibility of the Monte Carlo simulation. This prevents us from paying the extra cost of using increased precision math everywhere in the code.

1.2 A simple radiative transfer test problem

Consider an infinite medium radiative transfer problem [2]:

$$\frac{1}{c}\frac{de_\nu}{dt} + \kappa(\nu,T)e_\nu = \kappa(\nu,T)B(\nu,T), \tag{1}$$

where e_ν denotes the photon energy density with frequency ν. In this work the material opacity is assumed to have the form [2]:

$$\kappa(\nu,T) = \frac{2700}{\nu^3 T^{3/2}}\left(1 - exp(-\nu/T)\right), \tag{2}$$

where T is the material temperature, and $B(\nu,T)$ is the Planck function. This is coupled to the material via the equation of state:

$$Cv\frac{dT}{dt} = \int_0^\infty \kappa(\nu,T)e_\nu d\nu - \int_0^\infty \kappa(\nu,T)B(\nu,T)d\nu, \tag{3}$$

where $Cv = 0.5917aT_0^3 = 8.11 \times 10^{13}$ [erg-cm^{-3} keV^{-1}] is the material specific heat, a is the radiation constant and $T_0 = 1$ [keV] is the initial temperature. These coupled set of non-linear equation provide a good test base to verify reproducibility. The analytic steady state solution for the energy density can be written as:

$$e_\nu(t,r) = B(\nu,T_0) \tag{4}$$

because this is an infinite medium problem with no external source. The variable T_0 is the initial material temperature and r denotes a position in space. The analytic temperature distribution can be defined as $T(t,r) = T_0$. The numerical noise that arises from Implicit Monte Carlo solution approach and the non-linear feedback from the material proprieties make this problem difficult to reproduce in parallel, even though its analytic solution was easily defined.

The infinite medium problem was simulated using both a 1D and 2D mesh with constant face source equal to the initial material temperature. The units are defined such that the speed of light $c = 1$, the Plank constant

$k = 1$, the radiation constant $a = 1$, the Stefan Boltzmann constant $\sigma = \frac{ac}{4}$, and the Boltzmann constant

$$h = \left(\frac{8\pi^5 k^4}{15c^3 a}\right)^{\frac{1}{3}} = \left(\frac{8\pi^5}{15}\right)^{\frac{1}{3}}. \tag{5}$$

The 1D mesh was defined as $0 \leq x \leq 1$ using 100 equal spaced cells. The 2D mesh was defined as $0 \leq x \leq 1$ and $0 \leq y \leq 1$ using 100 equal spaced cells in both dimensions. A time step size of $\Delta t = 0.016$ was used. The initial material temperature was defined as $T_0 = 1.0$. These two test cases will be used in the remainder of the paper.

1.3. The consequences of parallelism for reproducibility

There are two modes of parallelism typically employed for Monte Carlo algorithms: domain decomposition and domain replication. In this work "domain" refers the section of the mesh on a processor. Domain decomposition is necessary when the mesh is too large to fit in the memory of one processor. It is also done to make problems run faster by bringing more computation resources to bear. To domain decompose the problem, we partition the grid and put sections on different processors. This necessitates moving particles between domains when they are tracked to domain boundaries.

Domain replication distributes work over multiple processors by performing a fraction of the set of particles on individual processors with identical spatial domains. The results are then summed together at the end of a time step. Domain replication is very useful with Monte Carlo algorithms because Monte Carlo particle histories are independent of one another over a single time-step. This allows the particle histories, which are the bulk of the computational cost, to be easily and efficiently distributed amongst multiple processors.

Two main issues are introduced when the number of domains can vary. The first is that the particles may not get the same pseudo-random number stream. The second is that the order of operations can change because particles interact with mesh zones in a different order, causing differences in the results arising from the fact that floating point addition is not associative.

1.4 Demonstrations of non-reproducibility

We will examine the 1D test case described in section 1.2 to demonstrate non-reproducibility of a Monte Carlo algorithm. We will compare two Monte Carlo simulations in which the particles use the same random streams but advance the particles in a different order, with a third simulation that uses a different random number stream. We will refer to the simulation using a different random number stream as the "fiducial" solution. We show that changing the order in which the particles are advanced, which changes the order of summation of energy deposition, eventually produces differences between the simulations that are of the same order of magnitude as the difference between simulations using entirely different random number streams. All simulations used the pseudo-random number generator described in [1].

We find the zone with the largest difference in temperature between the two simulations in each time step, and we plot the relative difference $|\Delta T|/T_0$ vs time step. (Since $T_0 = 1$ [keV], this quantity is the same as $|\Delta T|$.) This quantity is referred to as the "maximum relative difference" in the discussion below.

Each simulation was done twice, once using the c++ standard 64 bit `double`, and once using the non-standard 80 bit `long double` data type that is available on x86 architectures. The variables in the IMC code that were implemented as `long double` are described in section 1.1. When the 80 bit data type was used to sum a value, it was cast into an standard 64 bit `double` upon completion of the summation. For example, the energy absorbed and the energy emitted were tallied into 80 bit long doubles, they were then cast into doubles before taking the difference between the absorbed and emitted energies to update the temperature of the material. The calculations using the 80 bit data type were done to illustrate that adding a few more bits of precision to the sums does not guaranty reproducibility.

The simulations, seen in Figure 1, compares two serial (i.e. single processor) simulations of the 1D test case described in Section 1.2. A "forward" solution ran the source photons in the order they were created. A "reverse" solution was created by running the list of source photons in reverse order for every time step. Thus any difference in the results of the forward and reverse simulations is due to differences in the order of floating point addition when the particles deposit energy. The maximum relative difference in temperature was evaluated by comparing the forward solution to the reverse solution. The maximum relative difference between the forward solution and the fiducial solution, two simulations with different random number seeds, was also evaluated to illustrate the maximum error associated with the statistical deviations in the solution. Each of these problem were run on the 1D mesh, defined in section 1.2, with 10^4 particle histories per time-step.

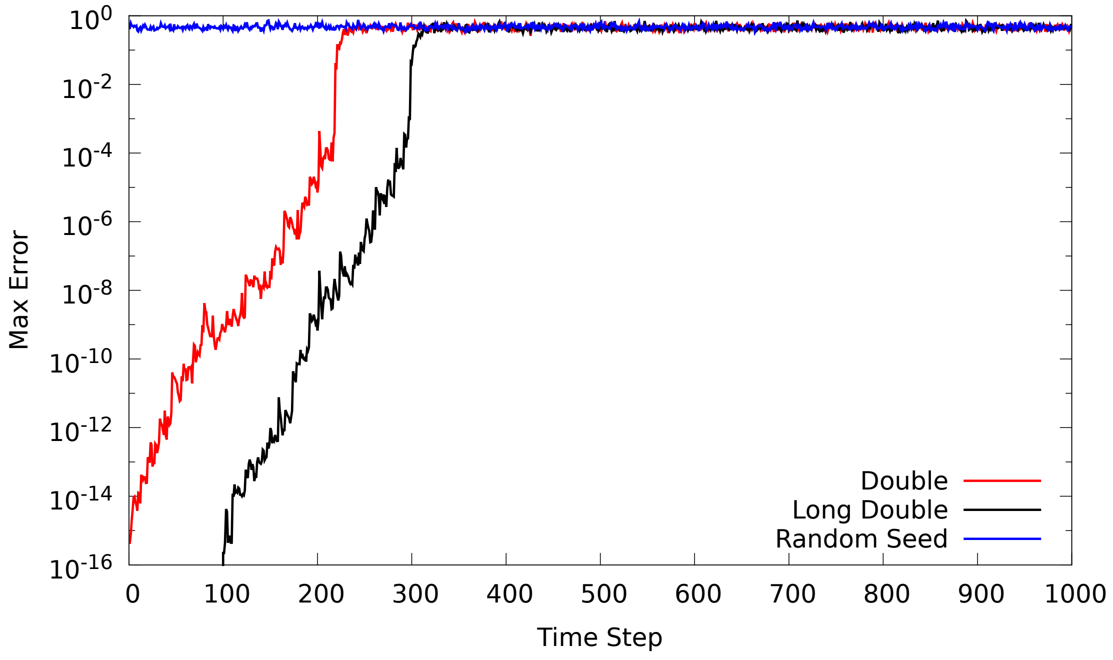

Figure 1. The maximum relative difference of the material temperature vs. time step for two different serial simulations, one with the photons in reverse order of their creation and another in the standard order. "Double" refers to using a standard double precision variable for all quantities with no extra rounding, and "Long Double" to using the non-standard `long double` data type.

2. ALGORITHMS THAT ACHIEVE ASSOCIATIVITY FOR FLOATING POINT ADDITION

In this work we investigate three algorithms that allow a reproducible answer for Monte Carlo simulations.

The first involves mapping from double precision values to 64 bit integer values. This method was described in [1]. The essential point of this method is doing sums of scaled integer values instead of double values, because integer arithmetic is associative. The drawback of this method is the potential loss of accuracy, and the possibility of overflow, in scaling double values to integer values and vice-versa.

The next two algorithms use data types other than `double` to accumulate sums, but cast the results back to a double precision value, rounding the extra precision, before using them. One method uses `__float128`, a 128 bit precision type available in GCC for x86 chips. The other uses a c++ class called `HPDouble`. This class contains 2 double precision variables, and implements addition in a way that keeps track of roundoff giving the data type approximately 106 bits of precision. This allows the sum of a series of doubles held in an `HPDouble` variable to be cast into a double result that is insensitive to the order of summation.

Each of these algorithms is described in more detail below.

2.1 64 bit integer algorithm

In [1], the non-associativity of double precision arithmetic was dealt with by using 64 bit integer values to do additions. Integer addition is associative, so the order in which values are added does not change the final answer. Using integers to hold tally quantities required that a conversion factor be calculated that mapped the maximum possible value of the double precision tally quantity to the maximum value of the 64 bit integer, which is $2^{64} \approx 1.84467 \times 10^{19}$. (In practice, the maximum value was set to $2^{63} \approx 9.22337 \times 10^{18}$ to allow a safety factor.)

This approach has two main draw backs. The first is the difficulty in determining the scaling factor which is used to map the double precision values to the integers. Using too large of a scaling factor causes the addition of small double precision values to be mapped to an integer value of zero. Too small of a scaling factor will cause large double precision values to overflow the integer range. The second draw back is that the overall accuracy of the sum is reduced below double precision, and the degree to which it is reduced is dependent on the selection of the scaling factor.

2.2 Extended precision double implemented in software

Extended precision algorithms are commonplace and have been investigated for use in improving parallel reproducibility [6]. The work of Robey et al. showed that extended precision algorithms can improve the accuracy of unordered sums, though small differences in the summations can still prevent reproducibility in coupled systems of equations [6]. Similarly arbitrary precision algorithms are widely published and available [7,8]. However their computational expense make them undesirable for use in most large multiphysics software packages such as Kull.

There are a variety of arbitrary precision arithmetic algorithms available in which extended numerical precision is achieved using a series of double values [7]. The goal in this work is to use enough precision that rounding the unordered sums to double precision will be bit for bit reproducible without significantly increasing the computation costs. We have implemented a high precision double value expansion, to be used for unordered sums. This high precision double extends precision to 106 bits of significand and is a reimplementation of the `double-double` data type defined in [9]. The high precision double algorithm is only implemented for addition. Multiplication and division are handled by casting the high precision double to a standard double.

The algorithm [9] for high precision double operators can be broken down into a series of floating point operator algorithms [7]. Two separate addition algorithms are used to expand the sum of two floating point numbers ($c = a + b$) into a two term expansion ($c = d + e \approx d$) [7]:

```
define two_sum(double a, double b)
{
  double value, error;
  double approx_b;
  value = a + b;
  approx_b = value - a;
  error = (a - (value - approx_b)) + (b - approx_b);
  return (value, error);
}
```

and

```
define quick_two_sum(double a, double b)
{
  double value, error;
  value = a + b;
  error = (b - (value - a));
  return (value, error);
}
```

Using these components, three primary operators need to be defined: the addition of a extended precision double value to a double value, the addition of two extended precision values, and the rounding from an extended precision value to a double precision value.

A high precision double c++ class called `HPDouble` was constructed that contains 2 doubles as private members (`value` and `correction`) which implements overloaded operators for addition. The addition of a `double` value (`A`) to an `HPDouble` value `B=(B.value, B.correction)` can be expressed as [9]:

```
define Accumulate(double A, HPDouble B)
{
  HPDouble C;
  (value, correction) = two_sum(A, B.value);
  correction += B.correction;
  (C.value, C.correction) = quick_two_sum(value, correction);
  return C;
}
```

where `C` is the resulting `HPDouble` value from `C=A+B`. The addition of two `HPDouble` values (`A` and `B`) can be expressed as [9]:

```
define Reduce(HPDouble A, HPDouble B)
{
  HPDouble C;
  (value, correction) = two_sum(A.Value, B.value);
  correction += A.correction;
```

```
    correction += B.correction;
    (C.value, C.correction) = quick_two_sum(value, correction);
    return C;
  }
```

The HPDouble is rounded to a double value by summing the correction and the value associated with it:

```
  define RoundToDouble(HPDouble A)
  {
    double C = 0.0;
    C += A.correction;
    C += A.value;
    return C;
  }
```

this ensuring the appropriate IEEE-754 rounding.

We note that the function Accumulate described above can be implemented in c++ using operator overloading as operator $+$, so that the accumulation of a double precision value B into an HPDouble value A can be implemented as $A = A + B$. Similarly, the RoundToDouble function can be implemented as a cast. These changes make using the HPDouble algorithm less intrusive in an already-existing code.

The algorithms described in [9] require the use of IEEE-754 behavior for double precision arithmetic. The reason is that the 80 bit registries on the x86 platform can round twice [10]. IEEE-754 behavior is needed for multiplication and division operators but does not seem to effect the accuracy of the summations which are rounded to double precision in this work. We found that improved precision for addition was obtained without setting the control word in the floating-point unit of the x86 processor. That is, we did not find it necessary to force our simulations to use IEEE-754 behavior. However discrepancies in underflow/overflow values as described by Shudo [10] could cause problems in simulations executed on different types of processors. The algorithms [9,10] transcribed in this work were specifically chosen rather than those presented by Robey et al. [6] because the reduce function that Robey presented produced slight difference in the lowest bit of the double value when rounded from extended precision.

We will illustrate the use of the software extended precision double algorithms described in this section by applying them to the calculation of the census energy in the IMC algorithm. (In the following, a superscript R denotes using a specified reproducible precision value rather than the standard double precision floating point value.)

In the IMC algorithm the census energy is defined to be the sum of the energy of all photons that reach the end of the time step. In a serial simulation the census energy can be defined for a single cell as:

$$E_\nu^R = \sum_p E_p(\nu) \tag{6}$$

where $E_p(\nu)$ denotes the energy of a photon p at frequency ν that reached the end of the time step. This sum is numerical evaluated with the Accumulate(E_ν^R, $E_p(\nu)$) function defined in section 2.2. The photon energy density for that cell can be defined as:

$$e_\nu = \frac{\texttt{RoundToDouble}(E_\nu^R)}{\Delta V} \tag{7}$$

where ΔV is the volume of the corresponding cell. The `RoundToDouble` function is defined in section 2.2. In a domain decomposition simulation, as discussed previously, the order of this sum can vary. In domain replication the individual partial sums on all replicas are summed via an MPI call at the very end of a time step such that:

$$E_\nu^R = \sum_j E_\nu^{R,j} = \sum_j \sum_p E_p^j(\nu) \tag{8}$$

where E_p^j is the energy of a photon which reached census on processor j. The sum over photons p for each processor j is evaluated using the `Accumulate(` $E_\nu^{R,j}$, $E_p^j(\nu)$) function and then the final energy is evaluated by summing the total energy on each of the processors ($E_\nu^{R,j}$) using the `Reduce(` E_ν^R, $E_\nu^{R,j}$) function. The tallied photon energy value becomes a source for the next time step of the problem. It can be seen that without associative proprieties for these sums the reproducibility of the problem will break down. In a similar way, the absorbed energy needs to be associative because it determines the material temperature (via Eq. 3) and, therefore, the emission source for the cell at the next time step.

These sums which require reproducibility are rounded to a predetermined precision at the end of every time step to prevent small difference from effecting subsequent time steps. The degree of rounding for the integer method is implicitly defined in the scaling it uses. A different degree of rounded was used for each of the fixed precision rounding approaches. All fixed precision rounding approaches where rounded to at least double precision to match the maximum precision for all other arithmetic in these simulations.

3. RESULTS

Here we compare the performance of the algorithms described in section 2. All of the three algorithms described in section 2 were able to make both the 1D and 2D IMC simulations reproducible when they were used for the sums described in section 1.1. The relative merits of the algorithms are execution time, the difficulty of implementation, and whether the algorithm required some problem-dependent input, such as the number of bits to shave.

The integer approach gives reproducibility, but suffers from a reduction in significant digits resulting from mapping energy from double precision variables to 64 bit integer variables and back. Since this mapping depends on the total energy in the problem, the precision of the integer method will vary from problem to problem.

The 128 bit float and `HPDouble` approaches avoid this problem dependence. The 128 bit float and the `HPDouble` precision approaches round at least 54 low order significand bits when they are cast to double precision. Converting from both 128 bit float and `HPDouble` to double precision has the advantage that the problem maintains 52 bits of significand, the same number of significand bits as the double precision data type. Rounding at least 54 low order significand bits will likely be enough to prevent any differences in the highest 52 significand bits for most problems.

3.1 Timing and accuracy

Table I shows run time and their associated standard deviations for each method in the 1D and 2D test cases using 10^7 particle histories per times step. The ”Double” and ”Long Double” test cases are not reproducible while all other test cases are. The 2D test case was decomposed on 64 processors and the 1D test case was domain replicated on 64 processors. All runs where performed on 2.8 GHz Intel Xeon processors with 2 GB of memory per processors.

Table I. Average simulation times [sec] for each summation type and the ratio of the run times for the double precision method compared to the other approaches.

	1D		2D	
Test Case	time [sec]	ratio	time[sec]	ratio
Double	706 ± 1	1.00 ± 0.00	848 ± 0	1.00 ± 0.00
Long Double	743 ± 1	1.05 ± 0.00	903 ± 34	1.06 ± 0.04
Integer Algorithm	705 ± 2	1.00 ± 0.00	923 ± 34	1.09 ± 0.04
`HPDouble`	776 ± 5	1.10 ± 0.01	948 ± 35	1.12 ± 0.04
128 bit Float	1702 ± 35	2.41 ± 0.05	2000 ± 1	2.36 ± 0.00

The key to gaining reproducibility is to eliminate the roundoff error in double precision addition. In [1] this was done by replacing the addition of double precision numbers with 64 bit integers which allows for exact reproducibility. The 64 bit integer approach rounds the significand in an dynamic way where the rounding is dependent on the span of values being transformed to integers during the summation. This work demonstrates how to obtain reproducibility by rounding the significand of the resulting unordered sums. The maximum number of significand bits that can be used are determined by the default floating point precision being used, which is 52 bits of significand for double precision in this work.

Statistical noise is created by the randomness used in the Monte Carlo approach and is an artifact of using a discrete number of Monte Carlo packets to estimate a continuous integral. The statistical noise is the random deviation of the numerical solution from the mean value. The analytic mean value is $\overline{T}(r,t) = 1.0$ for this infinite medium state state problem. Therefore, the statistical noise can be estimated using the standard deviation of the average material temperature from the analytic mean. The average material temperature for all the timed 1D simulations is $T_{1d} = 1.000 \pm 0.001$. The average material temperature for all timed 2D simulations is $T_{2d} = 1.00 \pm 0.01$.

4 Conclusions

We have described algorithms that can be used to make domain replicated and/or domain decomposed Monte Carlo photonics code produce the same answer, bit for bit, independent of the number of processors.

Two main issues are introduced when the number of domains can vary. The first is that the particles may not get the same pseudo-random number stream. The second is that the order of operations can change, causing differences in the results arising from the fact that floating point addition is not associative.

The first issue is eliminated by giving each particle its own random number generator state, and seeding these states in a manner that is independent of the number of domains. The second issue is dealt with in one of two ways. One is by by using integers, which are associative, to do addition. The other is to use rounding to remove numerical accumulated error in predetermined precision sums.

We chose to use a pseudo steady state problem which is very optically thick with a strongly non-linear opacity as a test base for this work. This was done in part because of the ease in evaluating the analytic solution. It was also chosen because it is a particularly difficult to get a reproducible answer to the problem. Being in thermodynamic equilibrium the net energy change from time step to time step should be exactly zero. As such, this problem had to do many subtractions of nearly equal numbers which is known to produce significant rounding errors for floating point arithmetic. This means that seemingly insignificant differences in unordered sums are greatly magnified.

Exact reproducibility was achieved for the integer, 128 bit float, and `HPDouble` approaches in every test case. While the double and long double approaches were not reproducible. It was shown that the accumulation of error for the double and long double precision approaches causes the solution to diverge from reproducibility because of the accumulation of numerical error in correlated sums. This is best illustrated by the accumulation of error when the particle order is reversed on a single processor (shown in Fig. 1).

Though standard double precision is not reproducible, it is pertinent to point out it is the fastest approach. The 128 bit floating point precision simulation was the only reproducible approach that was found to be significant slower than the others. The 128 bit floating point approach is also limited in implementation because it is only native to x86 processes.

We found that our high precision double, which was adapted from the double-double routines of the QD library [9], produces sufficient accuracy for double precision reproducibility in all tests cases presented. This software extended precision approach is particularly attractive because of its speed and platform independence as compared to the 128 bit floating point precision used in this work. Even though the integer approach presented by Gentile et al. [1] is the most robust reproducible method investigated in this work in that it guarantees reproducibility for all problems, the loss of overall double precision accuracy and the uncertainty of the degree to which the accuracy is lost makes it not an ideal solution for reproducibility. The best solution in almost all cases is to use the extended precision rounding discussed in this work.

Though this work found that rounding extended precision variables to double precision can be reproducible without strictly enforcing the IEEE 754 standard it should be done with caution, because of the previously stated issues with double rounding and under/overflow. Similarly it should be stated that even though rounding extended precision unordered sums will be reproducible in most practical applications, it does not strictly guarantee reproducibility for all problems.

ACKNOWLEDGEMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

REFERENCES

[1] N. A. Gentile, M. A. Kalos, and T. A. Brunner, Obtaining Identical Results on Varying Numbers of Processors in Domain Decomposed Particle, in Computational Methods in Transport (F. Graziani, Ed.), Springer-Verlag, Berlin (2006), vol. 1, p. 43.

[2] J. Fleck Jr. and J. Cummings Jr., An implicit Monte Carlo scheme for calculating time and frequency dependent nonlinear radiation transport, *Journal of Computational Physics*, **8**, 3, pp. 313–342 (Dec. 1971).

[3] N. A. Gentile, N. Keen, and J. Rathkopf, The Kull IMC package, Tech. rep., Tech. Rep. UCRL-JC-132743, Lawrence Livermore National Laboratories, Livermore, CA (1998).

[4] T. A. Brunner and T. A. Mehlhorn, A users guide to radiation transport in ALEGRA-HEDP, Tech. rep., Tech. Rep. SAND-2004-5799, Sandia National Laboratories, Albuquerque, NM (2004).

[5] T. A. Brunner and P. S. Brantley, An efficient, robust, domain-decomposition algorithm for particle Monte Carlo, *Journal of Computational Physics*, **228**, 10, pp. 3882–3890 (Jun. 2009).

[6] R. W. Robey, J. M. Robey, and R. Aulwes, In search of numerical consistency in parallel programming, *Parallel Computing*, **37**, 45, pp. 217–229 (Apr. 2011).

[7] J. Richard Shewchuk, Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates, *Discrete & Computational Geometry*, **18**, 3, pp. 305–363 (1997).

[8] The GNU MP Bignum Library, http://gmplib.org/ (2012).

[9] Y. Hida, X. S. Li, and D. H. Bailey, *Library for Double-Double and Quad-Double Arithmetic*, NERSC Division, Lawrence Berkeley National Laboratory (2007).

[10] K. Shudo and Y. Muraoka, Efficient implementation of strict floating-point semantics, in In Proc. of 2nd Workshop on Java for High-Performance Computing (in conj. with ICS 2000), (2000), pp. 27–38.